阿里云 | intel

第二届数据库大赛

Tair 性能挑战

“新手门槛” 战队答辩

主办单位：阿里云计算有限公司　英特尔（中国）有限公司

指导单位：中国电子技术标准化研究院

承办单位：阿里云数据库产品事业部　阿里云基础产品事业部和阿里云天池大赛

团队成员介绍

邓成杞

研究兴趣：机器学习、语音合成

https://github.com/KinglittleQ

唐小虎

研究兴趣：边缘计算、模型加速

https://github.com/tigert1998

姚璐

研究兴趣：机器学习

https://github.com/ClarissaYL

浙江大学 CAD&CG国家重点实验室

机器学习组 一年级硕士研究生

阿里云 | intel

目录

01 赛题背景和分析

02 AEP硬件特性

03 数据库设计

04 总结

阿里云 | intel

目录

01 赛题背景和分析

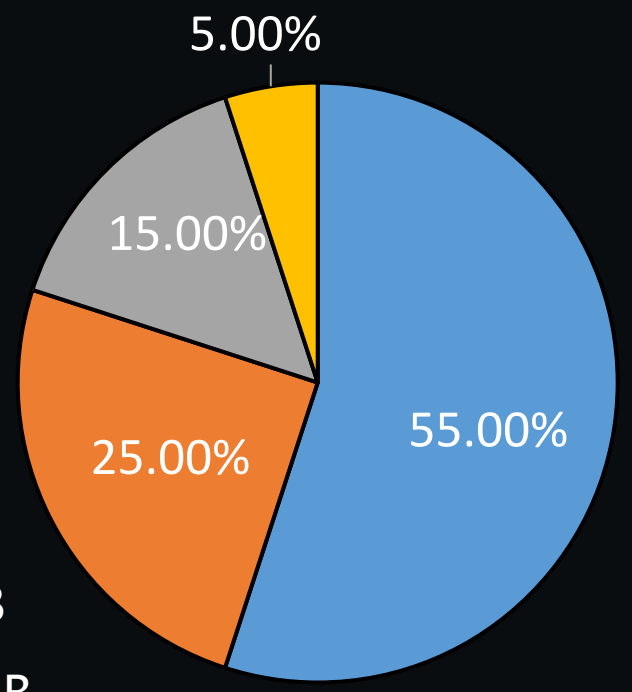
02 AEP硬件特性

03 数据库设计

04 总结

01 赛题背景和分析

- key 固定 16B，value 范围为 80-1024B。
- 性能测试包含两个阶段：随机写入、读写混合。
- 数据库需要在断电情况下保证 consistency。
- 读写混合阶段中 75% 是读，并且读操作中 75% 具有热点特性。
- 随机写入与读写混合阶段各会开16线程调用接口。



写入阶段的 value 比例

阿里云 | intel

目录

01 赛题背景和分析

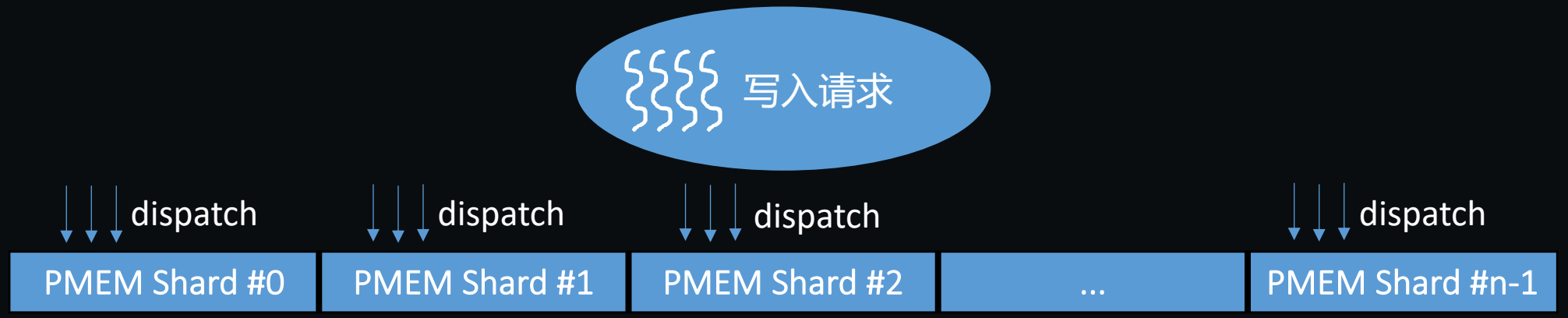
02 AEP硬件特性

03 数据库设计

04 总结

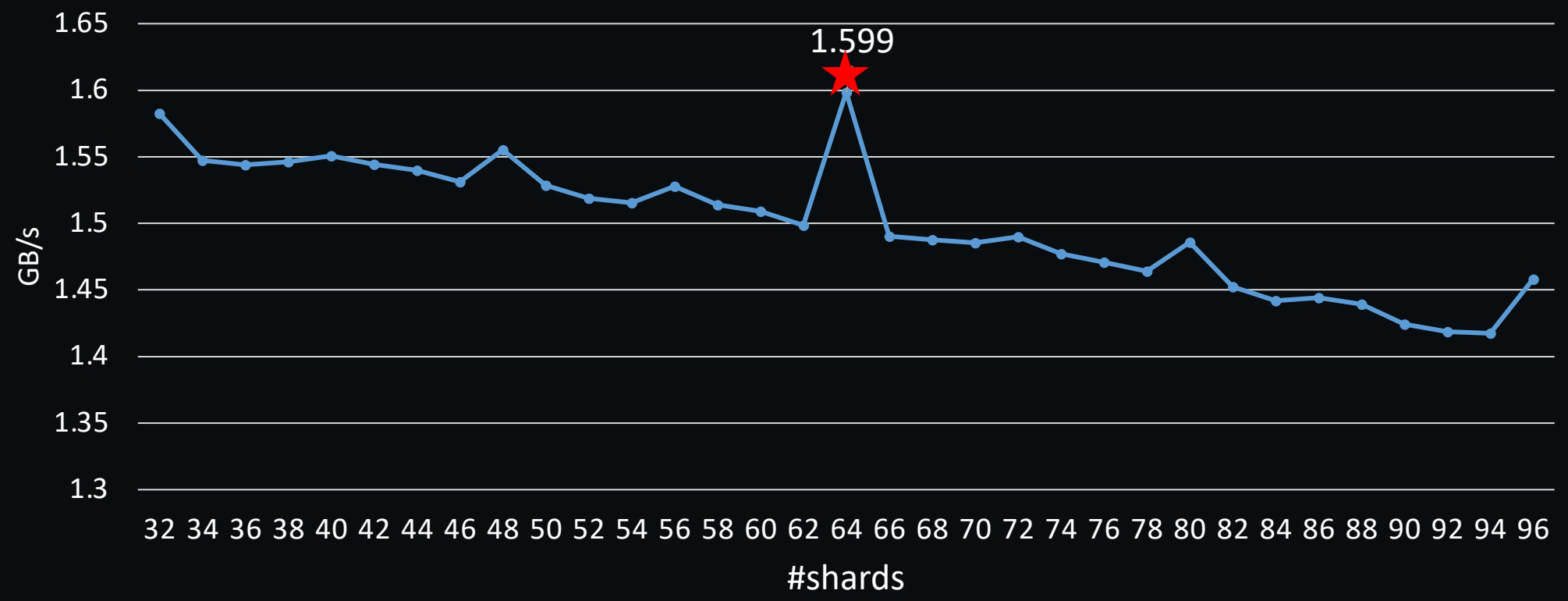
02 AEP硬件特性

- 赛题要求每次写入必须落盘。
- 256B 随机写入（ntstore）就可以达到 >2GB/s 的带宽。
- <256B 的随机/顺序写入带宽都很低。
- 分片顺序写入能有效提高不规整（非 256B） 写入的带宽。
- 写入地址、写入量对齐能有效提高带宽。



02 AEP硬件特性

- Benchmark 结果表明：分片的数量在64时吞吐量最大。
- 下图：按照题目分布随机生成 96 – 1040B 的 record，用16线程分片顺序写入（ntstore）pmem 时的吞吐量，record 的首地址和写入量会被对齐到32。横轴是分片数量。

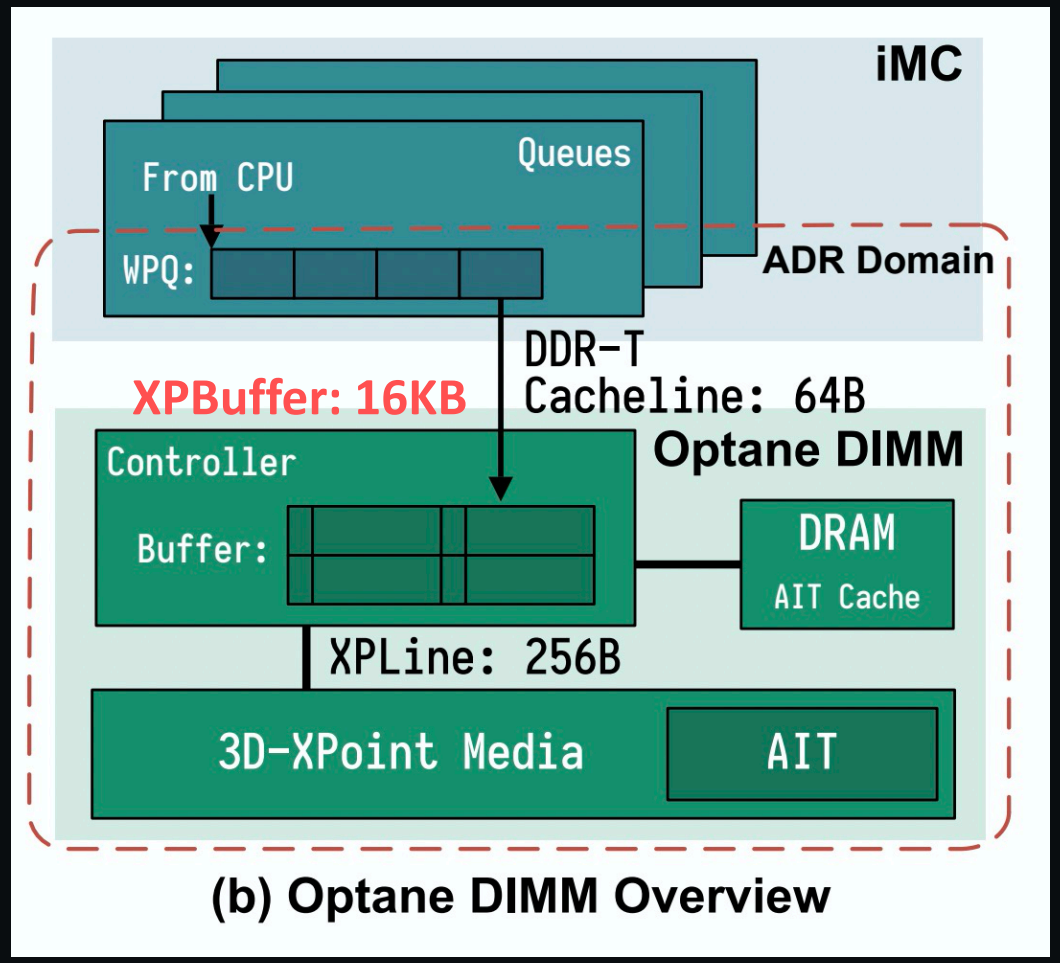


02 AEP硬件特性

- Benchmark 结果表明：record 的写入地址和写入量均对齐到64时吞吐量最大。
- 下表：按照题目分布随机生成 96 – 1040B 的 record，用16线程分64片顺序写入（ntstore）pmem 时的吞吐量。纵轴：写入地址的对齐量；横轴：record 写入量的对齐量。单位：GB/s。

addr \ size	32	64	128	256
32	1.58895			
64	1.722522	1.791549 ★		
128	1.529643	1.60189	1.64475	
256	1.352903	1.417234	1.427518	1.452822

02 AEP硬件特性



(b) Optane DIMM Overview

解释：

- 分64片：XPBuffer / XPLine = 64
- 写入地址和写入量对齐到64B：DDR-T Cacheline = 64B

阿里云 | intel

目录

01 赛题背景和分析

02 AEP硬件特性

03 数据库设计

- 索引设计
- Record 结构设计
- 分配器设计
- Get/Set/恢复流程

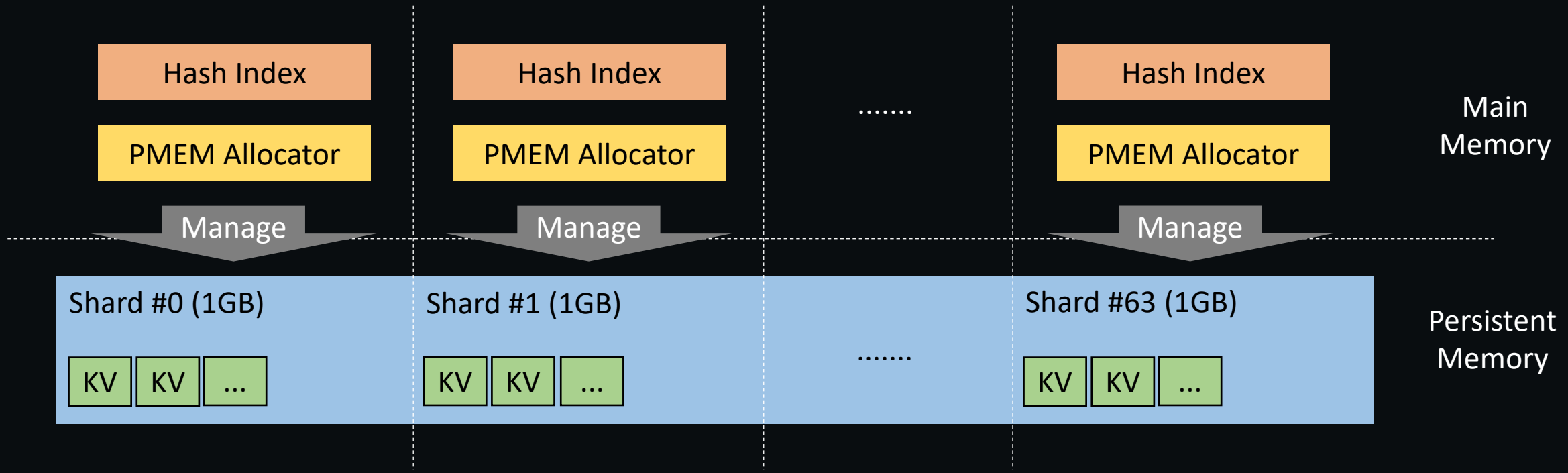
04 总结

Guidelines

- 把 key 和 value 连接为一个 record 整体读写以彻底消除对 pmem 的短更新。
- 把 record 按照 key 的 hash 值均匀分布到64片上处理。
- record 的首地址和写入量都必须对齐到64B。
- 全部使用 lockfree 的数据结构以消除context switch开销。

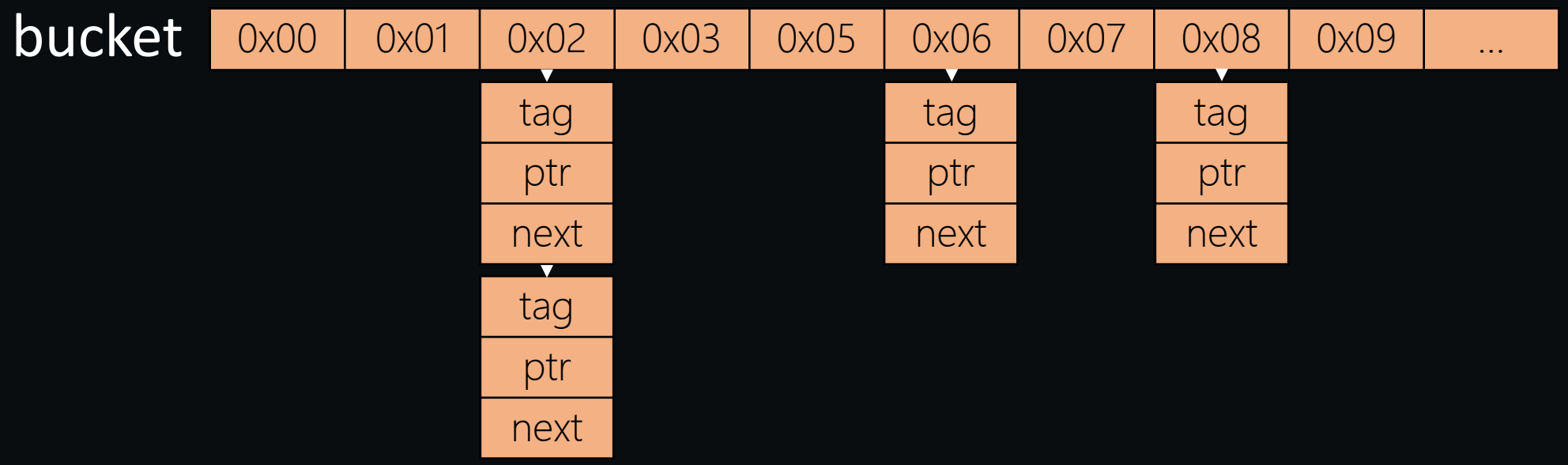
03 数据库设计 – 整体架构

- **Set**： 在 pmem 中 allocate 一块内存写入 KV 并修改 index，之后回收 旧 KV 的内存块。
- **Get**：查询 index 并返回结果。

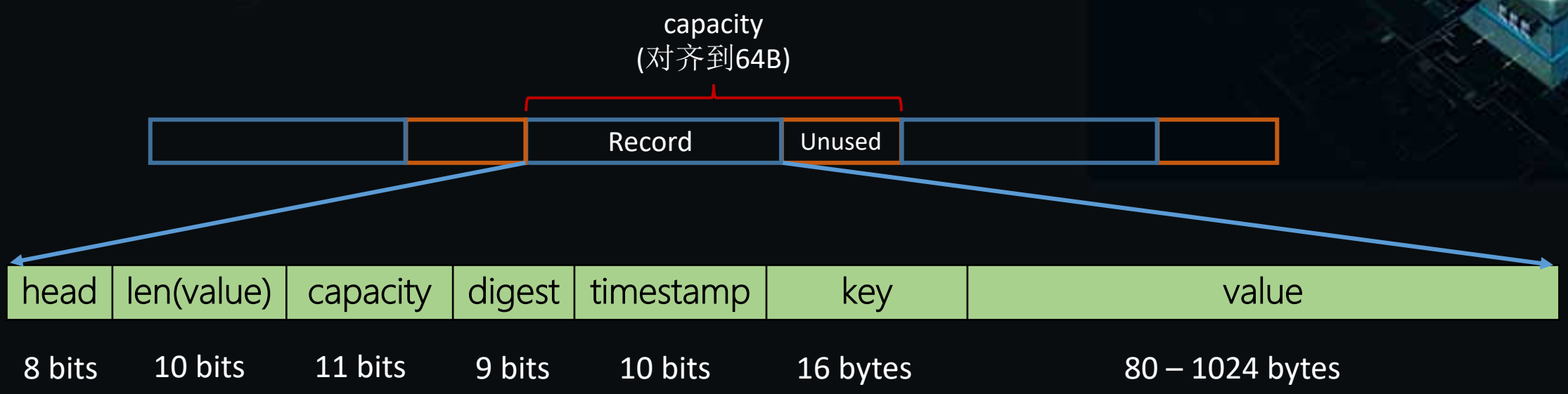


03 数据库设计 – 索引设计

- 索引记录了从 key 到 record 地址的映射。
- 因为全部是点查询，所以使用常数复杂度的 hash 索引。
- hash 值的前8位留作 tag，剩下的位数用于定位 bucket。
- tag 的目的是在不需要访问 pmem 的情况下，快速过滤同一个 bucket 内绝大部分的 key。



03 数据库设计 – Record设计



- head 是一个常数标识符，标识了 record 的首字节。
- len 标识了 value 的长度。
- capacity 标识了 record 实际占用的 pmem 空间。
- digest 是 record 的校验和，用于在数据库恢复时确认 record 是否完整，保证原子性。
- timestamp 存放了 record 的版本号。

03 数据库设计 – 分配器设计

- allocator 会将过时的 KV record 回收起来用于后续分配。record 中的 capacity 域标识了需要回收的长度。

- 分配请求会被 allocator 对齐到64B。

- allocator 的两种分配策略：

  1. PMEM未写满（顺序分配）：直接在已用 pmem 的末尾分配新的内存区域。
  2. PMEM写满后（碎片化）：在内存回收池内寻找一块合适（best fit）的内存块，并将多余的部分再回收到内存回收池中（ e.g. 需要 128B 但是分配到了 256B 的内存，多余的 128B 会被继续回收）。

03 数据库设计 – Get流程

Get (key)：

1. 计算 key 的 hash 值，前8位为 tag，剩余位数用于定位 bucket。
2. 遍历 hash bucket 以查询 key 是否存在，比较 key 时先比较 tag（1 byte）是否相同，如果相同则逐字节比较key（16 byte）。
3. 返回对应record的value。

03 数据库设计 – Set流程

Set (key, value) :

1. 分配 pmem 内存块

2. 查询 index 内是否存在 key

   1) 如果 index 内已经存在该 key，则记该 key 的 timestamp 为 t
      ① 构造 t + 1 的 record 并写入pmem
      ② 如果 index 内 key 已经被修改了则 goto step ①
      ③ 更新 index
      ④ 回收过时的 record

   2) 如果 index 内不存在该 key
      ① 构造 timestamp 为0的 record 并写入pmem
      ② 如果 index 中存在该 key，则 goto step 1)
      ③ 在 index 中插入 KV

阿里云 intel

03 数据库设计 – Set流程

Set (key, value)
（示例伪代码）：

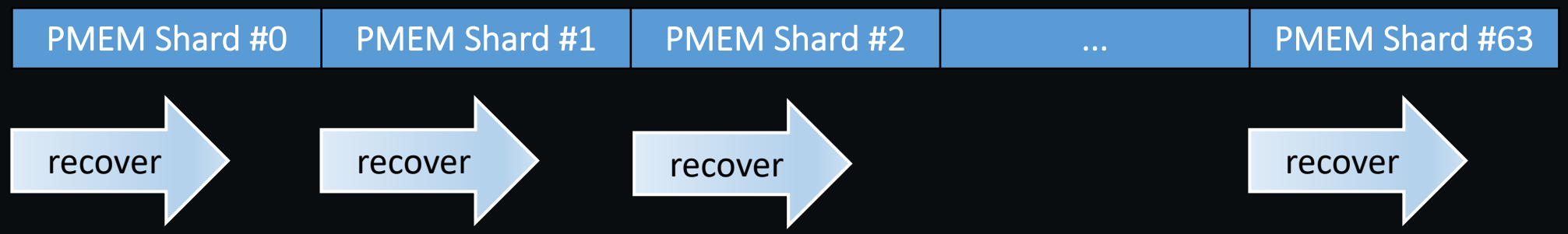
```
// 在pmem中分配内存
auto pmem_ptr = Allocate(record.size());
// 将record写入pmem
WritePmem(pmem_ptr, record);
// 查找hash index
auto bucket = FindBucket(buckets, key);
auto node = FindNode(buckets.nodes, key);
if (node != NULL) {
    // 更新
update:
    do {
        auto ptr = node.ptr.Load();
        pmem_ptr.timestamp = ptr.timestamp + 1;
        // 原子更新
    } while (!node.ptr.CompareAndSet(ptr, pmem_ptr));
    // 内存回收
    Deallocate(ptr);
} else {
    // 插入
    pmem_ptr.timestamp = 0;
    if (!bucket.Insert(pmem_ptr)) {
        // 已经被抢先插入，尝试更新key
        goto update;
    }
}
```

03 数据库设计 – 恢复流程

恢复流程

1. 对每个 pmem 分片顺序扫描恢复。
2. 如果 record 首字节正确而且能通过校验和则插入 index。
3. 如果 index 中存在该 key 并且 timestamp 比恢复出来的更小，则更新 index。

优势：Record可以灵活地存放在PMEM的任意位置



阿里云 | intel

目录

01 赛题背景和分析

02 AEP硬件特性

03 数据库设计

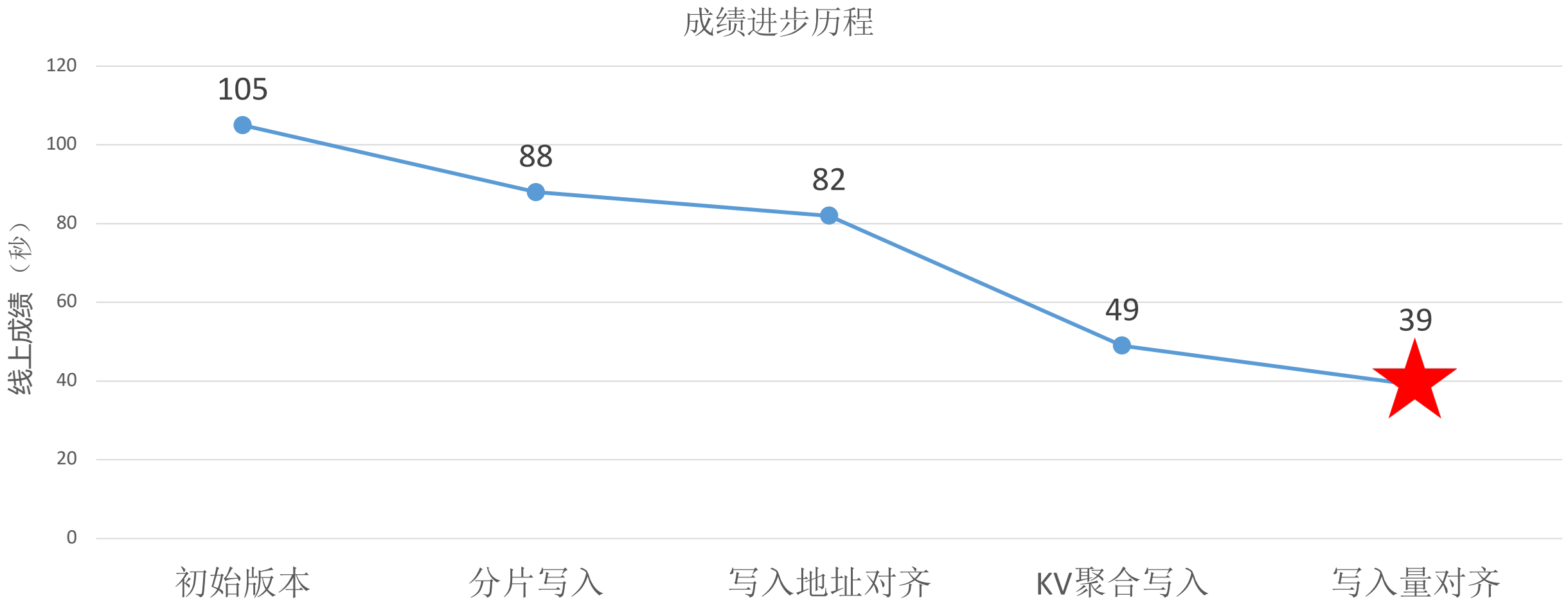
04 总结

阿里云 | intel

04 总结

- IO
  - 分片顺序写入能有效提高不规整（非 256B）写入的吞吐量。
  - 写入地址、写入量对齐能有效提高吞吐量。
- Record
  - 把 key value 聚合在一个 record 整体读写以消除对 pmem 的短更新。
  - 利用校验和保障写入 KV 的原子性。
  - 使用 timestamp 确保恢复出的 KV record 是最新的。
- 索引
  - 在 hash index 内维护key的 tag 以高效判断key是否相等。
- 内存分配器
  - allocator 将分配的多余的内存继续回收起来以提高利用率。

04 总结


成绩进步历程
线上成绩（秒）
120
100
80
60
40
20
0
105
88
82
49
39
初始版本
分片写入
写入地址对齐
KV聚合写入
写入量对齐

04 总结

最优成绩	时间
数据库初始化	~24s
写入阶段	27.144s
读写混合阶段	12.512s
总成绩	39.656s

阿里云 | intel

第二届数据库大赛

Tair 性能挑战

“新手门槛” 战队答辩

主办单位：阿里云计算有限公司　英特尔（中国）有限公司

指导单位：中国电子技术标准化研究院

承办单位：阿里云数据库产品事业部　阿里云基础产品事业部和阿里云天池大赛